# Custody Transfer for Reliable Delivery in Delay Tolerant Networks

Kevin Fall, Wei Hong, and Samuel Madden

IRB-TR-03-030

July, 2003



Intel Research Berkeley

# Custody Transfer for Reliable Delivery in Delay Tolerant Networks

Kevin Fall
*kfall@intel-research.net*

Wei Hong
*whong@intel-research.net*

Samuel Madden
*smadden@intel-research.net*

Intel Research, Berkeley
Berkeley, California

***Abstract*— We investigate the *custody transfer* mechanism proposed for enhancing reliability in delay-tolerant networks. This mechanism, which utilizes hop-by-hop transfer of reliable delivery responsibility, shares many features in common with a database transaction. By considering custody transfer in this light, we observe that it can cause the creation of duplicate message fragments within the network that ordinarily may pose no significant problem, but can be cause for concern if in-network processing and data fusion are employed. We also extend the DTN architecture with the concept of a transaction abort, and show how it can be used for helping to free network resources being consumed by fragments that remain after completion of a previous message transfer.**

## I. Introduction

The delay-tolerant networking architecture proposes a store-and-forward message-oriented routing overlay as an approach to deal with problems of disconnection, high latency, and heterogeneity in challenged internetworks [1], [2]. One of its suggested components, called *custody transfer*, offers a way to enhance end-to-end reliability in these networks by moving the responsibility for reliable delivery of a message toward its ultimate destination. Applications may request an (optional) end-to-end acknowledgment in addition to the enhanced reliable delivery provided by custody transfer.

In this paper, we further describe the operation and architectural implications of custody transfer, indicating an analysis of custody transfer as a database-style transaction. We first review the concept of custody transfer in DTNs and its approach to reliability and implications for congestion. We then consider details of the custody transfer mechanism itself, and note its similarity to database transactions. We conclude with its potential impact on the creation of duplicate messages within the network and how such duplicates can affect the operation of functions executed within the network itself that operate on queued data.

## II. DTN and Custody Transfer

A delay tolerant network consists of a directed graph $G = (E, V)$ where the set of directed edges $E$ are derived from a list of *contacts*. A contact describes a link's tail and head vertex, existence interval (from the tail's point of view), plus its (constant) capacity and latency during the interval.[1] An edge $e = (t, h)$ is placed in the set $E$ if $t$ and $h$ ever appear in a contact.

The set of vertices $V$ consist of store-and-forward message routers which may optionally provide *custody transfer*. Accepting a message with custody transfer amounts to promising not to delete it until it can be reliably delivered to another node providing custody transfer (or to the message's destination), to the best of the ability of the forwarder. Nodes holding a message with custody are called *custodians*. Ordinarily, there is a single custodian for a message (called *sole custody*), but in some circumstances more than one custodian owns a message or message fragment (called *joint custody*, see below). Applications optionally request custody transfer to be performed on a per-message basis, and are delivered a *custody acknowledgment* when their host system has been able to move the message to one or more other nodes that are willing to take custody for it. In particular, the custody acknowledgment is not an end-to-end acknowledgment, but instead indicates that the responsibility for end-to-end reliable delivery has been delegated to some other party apart from the sending node.

Although routers may offer custody transfer, this decision is at their own discretion and may be subject

[1] In cases of variable capacity or delay, we assume a piecewise constant approximation is adequate, given sufficiently small time intervals.

to change. In some circumstances a node that had previously been accepting messages and custody for them may cease accepting custody (while continuing to accept messages) if local node resources become substantially consumed. It may resume accepting custody when resources become more plentiful. It may also make individual decisions as to whether to accept a particular incoming message (and custody for it) based on routing, message priority or size, security, maximum message lifetime, and local policy constraints. Algorithms using only local state for this decision are under investigation.

Messages may be fragmented or aggregated as they traverse the network. Fragmentation may be employed by a sending node in circumstances where a contact may not be of sufficient volume to move an entire message to its next hop. Another form of fragmentation, called *reactive fragmentation*, may be employed by a receiving next-hop node if a link should become inoperative while a message is transferred (and the receiver is able to conclude at least a portion of a message has been received correctly). Given that communication opportunities in DTNs may be occasional and valuable, whenever messages can even be *partially* moved over an intermittent link successfully, it is valuable to do so and to not have to re-start anew with subsequent contacts. Re-assembly of fragments into complete messages (or larger fragments) is possible when multiple fragments for the same destination together await an outgoing communication opportunity. If reassembly of fragments does not occur in the network, it is accomplished at the end (receiver).

One of the advantages of custody transfer is to allow an an endpoint to free storage resources comparatively quickly—when a custody transfer acknowledgment arrives. Given that custody transfers are likely to take place with comparatively nearby nodes, the round-trip-time to complete a custody transfer is expected to be significantly shorter than an end-to-end round trip. For end nodes responsible for collecting data (e.g. remotely deployed sensors), the ability to free buffers quickly can be of great importance.

## III. Custody Transfer and "the End-to-End Principle"

As suggested in [1], the custody transfer mechanism may appear to be in contradiction to a design principle called the end-to-end argument [3] which states that placement of functionality at low layers of a system may be redundant and not properly address the true needs of applications (the final arbiters of what functionality they require). It also argues that functions placed at low layers of the system should only offer performance enhancements. In consideration of this, we must focus on where the application is placed, and if its needs could adequately be conveyed to another location (i.e. other than at the sender). If this is possible, then we may accept such a "movement of the application" as consistent with the argument.

A distinct but related argument often arising in the context of Internet discussions is the principle of *fate sharing* [4]. In essence, this principle states that one should avoid placing critical (connection) state in the middle of the network, and strive to place it only at the end stations. If end stations fail, it is reasoned, any network connections between them would as well, so connection robustness is enhanced by placing this critical state only at endpoints. In the DTN case, we assume that the network's longevity may exceed that of end nodes, and that in general it can be assumed to have comparatively reliable storage. Furthermore, in a case where no end-to-end network path may ever exist, the concept of an end-to-end connection is not present in the same way as it is for the Internet. Thus, it seems inherently problematic to apply the principle of fate sharing to DTNs. In some extreme cases it is worth noting that an end-to-end round-trip time might actually exceed the sending node's effective lifetime.

## IV. DTN Congestion

In a DTN router, the resources that may receive contention for access are storage at each node, and use of link bandwidth when contact opportunities arise. If we assume that allocation of link bandwidth is a local assignment problem with straightforward solutions, the remaining problem is to handle buffer allocation. While most packet networks can simply discard packets when volatile memory is exhausted, DTN messages received consume permanent storage, and generally cannot be safely discarded if custody has been accepted for them. The only available mechanisms to safely re-use the storage associated with a message for which custody has been accepted is either to move the message to another custodian, or delete the message because its application-specified expiration time has been reached. Other messages (without custody) can be discarded as required, and generally operate best-effort.

### A. Head-of-line Blocking

If a node with limited storage is acting as custodian for many messages, it may have little ability to accept addi-

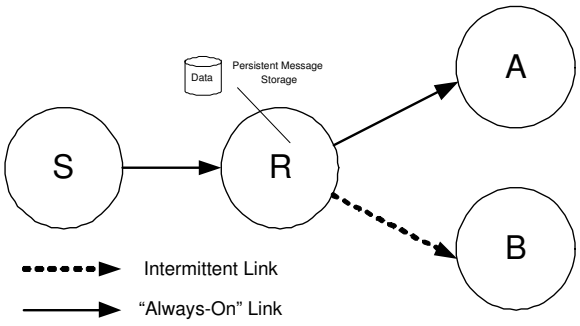


Fig. 1. A simple DTN topology that could cause head-of-line blocking. $R$ acts as a custodian for traffic from $S$ to $B$ awaiting the $(R, B)$ contact to open. Message traffic destined to $A$ from $S$ may be unable to transit $R$ even if the $(R, A)$ link is always available.

tional traffic. Furthermore, if it is naïvely implemented, it may be unable to forward traffic that only needs to pass through it. Figure 1 illustrates a case where this problem can arise. Here, router $R$'s persistent storage is filled by messages originating at $S$ and destined for $B$. No $R \rightarrow B$ contact is currently available, yet the $R \rightarrow A$ link is always available. Thus, even though $R$ provides "always-on" connectivity between $S$ and $A$, traffic headed from $S$ to $A$ may be unable to flow.

A more careful implementation of $R$ may alleviate this problem. Three strategies (at least) may be employed: reservation of persistent storage to non-custody transfer messages, in-memory store-and-forward, and "cut-through" in-memory routing. If the reservation strategy is adopted, the amount of storage to reserve becomes the primary question, but this value is highly dependent on the dynamics of topology, link bandwidth, maximum available storage and expected sizes of messages.[2] The in-memory approach is limited to the amount of available volatile memory, and is subject to loss upon node outage (which could negatively affect end-to-end delivery probability). The cut-through strategy is only applicable to always-available links, but has the advantage of minimizing overhead at the router (thereby reducing latency) and using less overall memory resources.

Given the constraints, an individual node must be careful when agreeing to accept custody for messages, as it would appear careful management of storage resources is required (especially when disconnection may be prevalent). While this problem remains to be fully investigated, it may prove worthwhile to look to a game-theoretic approach. A local decision to accept custody for a message mimics a decision to purchase a perishable commodity, wherein a larger message corresponds to a higher price, and time to next contact corresponds to liquidity.

[2]More experience is required to postulate a reasonable allocation scheme in this case.

## V. Semantics of the Transfer Operation

The purpose of a custody transfer is to move the responsibility for reliable delivery of a message from one node to another, starting at the sender and completing at the final destination. Messages (and message fragments) indicate their current custodian in the DTN protocol header, and this field is updated as custody of the message is moved from custodian to custodian. The ability to transfer fragments (created by the different types of fragmentation operations described above) complicates this situation somewhat: an end-node considers a message to have been successfully delivered with custody when all of its fragments have been transferred to one or more custodians. Custody transfer is typically accomplished when a node has a contact available to a next-hop router offering custody transfer. Note that the location of routers which provide custody transfer could have an effect on the routing computation, as a path that contains one or more routers with custody transfer capability may be preferential to a (shorter) path that does not.

### A. Custody Transfer as a Transaction

The end-to-end custodial delivery of a DTN message can be viewed as a single database-style ACID transaction. Because of the potential long delays in a DTN, this is a so-called *long-lived* transaction in the database literature. Long-lived transactions are inefficient to run as a single transaction because large long-lived transactions cannot take advantage of partially-completed work. It is therefore more efficient to run a long-lived transaction as a *nested* transaction, in which individual sub-transactions can be committed, in order to checkpoint forward progress (or rolled back in case of local failures). An interesting question for many long-lived database transactions is how to efficiently partition them into sub-transactions. This transaction partitioning problem is analogous to the problem of assigning custody to fragments of DTN messages.

For the end-to-end delivery of a DTN message, the finest-grain partitioning is to treat each fragment traversing each single hop as a transaction. Coarser granularity partitionings are possible: multiple message fragments across multiple hops. The choice of granularity is influenced by the overhead involved in using a finer granularity partitioning versus the probability of delivery

across multiple hops. If the network is able to deliver message fragments over multiple hops with few errors, it is advantageous to use a coarser granularity partitioning.

### B. Distributed Commit Protocol for Custody Transfers

The custody transfer of fragments from node $A$ to node $B$ is essentially a distributed commit of the sub-transaction between $A$ and $B$. It can be implemented by the standard 2-phase commit protocol [5] with node $A$ as the commit coordinator. Basically, $A$ sends fragments to $B$; $B$ sends back an acknowledgment to $A$ after storing the fragments in persistent storage. Upon receiving $B$'s acknowledgment, $A$ deletes the fragments from its storage and sends a "custody transfer complete" message to $B$. Finally, upon receiving $A$'s completion message, $B$ assumes custody of the fragments it received from $A$. As we will show, this protocol does not always work in DTNs because in some cases it is unable to complete.

### C. Distributed Abort

In considering custody transfer as a transaction, we are drawn to the concept of a transaction abort. As such, we suggest augmenting the DTN architecture to allow the sender of a message to recall the message (or fragments of it) after it has been sent. This constitutes a distributed abort of the transaction. The simplest way to support this is to send an abort message towards the destination. Along the way, each intermediate node will drop any fragments of the aborted message upon seeing the abort message. If fragment routing paths do not change drastically over time, this strategy is likely to be effective. Once the destination receives an abort message, it can notify the application of the cancellation. If the application has not yet consumed the aborted message, then the message and any of its constituent fragments may be simply discarded without involving the application.

Another case for distributed abort is for pro-actively suppressing duplicate fragments in the network. As will be shown in the next subsection, duplicate fragments can be common in DTN. Once the destination receives a fragment that potentially has duplicate copies in the network, the destination can send an abort message to all the intermediate nodes that might own a copy of the fragment. This can be accomplished either directly (if the destination is able to ascertain the path by which a message arrived) or indirectly by sending an indication back to the source of the message requesting it to issue an abort for the fragment[3].

### D. Network Partitions

The custody transfer protocol proposed above cannot provide transactional guarantees in the face of certain kinds of permanent network disconnections. Consider a sender $A$ and a receiver $B$, exchanging a message fragment $f$. Once $f$ has been transferred to and acknowledged by $B$, $A$ will send an acknowledgment indicating custody transfer is complete. If $A$ and $B$ are disconnected at this point, before $B$ receives the acknowledgment, $A$ will have ceded custody of the message and $B$ will not be able to verify this fact. The commit protocol could be extended with a third phase, such that $A$ will not cede custody until $B$ acknowledges the previous acknowledgment – however, this protocol is susceptible to partitioning in the same manner as the previous approach if $B$'s acknowledgment is lost.

Of course, in many situations, such network partitions will not be permanent – and so these protocols typically involve timeouts and retries at various points, with the ability to reconcile the state of a transaction when the connection is reestablished. However, there will certainly be situations in delay tolerant networks where a receiver or transmitter simply wanders away, never to be heard from again. Such partitionings lead to potential semantic problems, as we will see.

### E. Availability vs. Consistency: The Rise of Duplicates

Our problem is related the CAP theorem [6] posed by Fox and Brewer – no network can be simultaneously consistent, available, and tolerant to partitions. Following the logic of this theorem, when a delay tolerant network is subject to this kind of disconnection, we have two choices: either sacrifice availability by holding the message at $A$ until it comes into contact with $B$ again, or sacrifice consistency at $A$ by trying to transfer custody of $f$ to some other receiver, possible leading to multiple nodes that believe they are the custodian of $f$ and the possibility of duplicate fragments being transmitted.

In a traditional networking environment, the latter solution (possibly sending duplicates) would likely be the preferred solution, as duplicate elimination is considered relatively easy to do at the transport layer using sequence numbers. Duplicates also arise quite frequently in some routing protocols used in intermittently connected and high-delay networks, such as epidemic and gossip based

[3]This may involve piggy-backing this indication on the the end-to-end acknowledgment message (if enabled).

protocols [7], [8], but even here duplicate elimination is still considered straightforward.

## VI. Relationship Between Duplicates and In-Network Processing

We assert that, in many delay tolerant networks, duplicates may pose a larger problem: they hinder the ability to partially process data within the network. In-network processing is seen as desirable because it can dramatically reduce bandwidth requirements. Unfortunately, as we will see, if data is coarsely aggregated within the network it can be difficult to detect or eliminate duplicates, which can lead to incorrect answers.

In-network processing has been proposed in a number of delay-prone environments. For example, in sensor networks, bandwidth is generally scarce, especially at the edges of the network, and thus doing some fusion or aggregation of sensor readings as data is routed is potentially beneficial. A number of papers [9], [10], [11] note the benefits of in-network aggregation, citing order-of-magnitude or greater bandwidth reductions for some classes of operations given particular network topologies. Similarly, when moving data between different classes of networks (e.g. the Internet and GPRS), it may be useful to transcode or downsample data items, sometimes in a non-deterministic way, as when dithering an image.

If the network cannot guarantee duplicate-free semantics, some in-network operations might produce incorrect answers: consider a sensor network attempting to compute an average over the readings from a number of sensors. If one of these readings is duplicated, it will obviously skew answers. We call such operations *duplicate sensitive*. Of course, some in-network operations are *duplicate insensitive* – computing a minimum of a set of readings, for example, has this property.

Thus, we have seen that, unless we wish to sacrifice the availability of our network, duplicates may arise in disconnection-prone delay tolerant networks. Furthermore, because many such networks may wish to perform in-network computation, duplicates can be more problematic than in traditional networks. In the next section, we examine possible techniques for mitigating the overhead of duplicate elimination.

### A. Alternatives to Explicit Duplicate Elimination

One option for handling duplicates would be to add explicit duplicate elimination to duplicate sensitive operations. However, in many cases, this will completely eliminate the benefit of in-network processing. For example, consider a sensor network organized into an ad-hoc routing tree that is collecting the average of some physical attribute $a$ over all of its nodes. In the simple in-network protocol proposed in TAG [10], every node transmits a $\{sum, count\}$ pair where $count$ is the number of devices in its locally rooted subtree and $sum$ is the sum of the value of $a$ over all of those nodes. This has the benefit that only a pair of values are transmitted from every node on the tree, dramatically reducing the bandwidth requirements in a large network. Unfortunately, to add duplicate elimination to this approach, each of these pairs would need to be extended with a list of sensors that contributed to the sum and the amount of their contribution, effectively requiring the entire set of readings to be transmitted up to the root of the network.

We propose a simple alternative to explicit duplicate elimination: when a delay tolerant network chooses to attempt to transfer a fragment to another node, it may assert *joint custody* over that fragment. This means that, if the node cannot determine that custody transfer was successful (due to a partition), it reserves the option of attempting to transfer the fragment to a different recipient. A node that initially asserts joint custody may grant *sole custody* to the recipient once the two phase commit protocol has successfully completed. However, if this conversion to sole ownership is not done, or if the notification of sole ownership is lost, any subsequent forwarding of a joint custody fragment must include a tag indicating the fragment is in joint custody. The presence of this joint custody tag means that other nodes may expect to see duplicates of this fragment, and should compensate for this when doing in-network processing.

For example, we can attach to our $\{sum, count\}$ records a list of the joint-custody values and their contribution. When combining two such records together, duplicates can be easily eliminated. We expect that the size of these joint-custody lists will be small, since relatively few fragments should experience partitioning, and thus we are still able to perform in-network processing despite the presence of a few duplicates.

There are several open policy issues related to joint custody. First, the notification of sole custody introduces an extra message indicating that the sender has ceded joint custody along every hop in the network. In some cases, the cost of sending this extra message may outweigh the benefit of duplicate elimination, especially if the in-network processing is duplicate insensitive or no in-network processing is being done. Second, it is unclear how aggressive the network should be in forwarding notifications of sole custody; for example, if a node receives sole custody after it has already

forwarded a fragment in joint custody it could attempt to forward this sole custody notification as well. A node is, however, always free to suppress the notification and forwarding of sole custody, as more aggressive duplication elimination will never affect the *correctness* of in-network processing.

## VII. Conclusions

We investigate the custody transfer mechanism proposed for delay tolerant networks, with particular emphasis on its implications for congestion management and the semantics of its protocol operation. The congestion management problem can lead to a form of head-of-line blocking, and several techniques are available to handle the problem (at least in part). We discuss the relationship of this protocol to traditional transaction commit protocols, and observe that the problem of network partitioning can create duplicates that pose troubles if in-network message fusion is to be utilized. We propose the concept of joint custody to mark messages that may have been duplicated, which can help to ameliorate this problem. We believe the techniques we propose are an important first step for incorporating custody transfer techniques in delay tolerant networks, especially those operating in constrained environments where in-network data processing can offer significant performance gains.

## Acknowledgment

The custody transfer mechanism was proposed initially in [12]. The authors would like to thank the authors of that document, especially Bob Durst, Keith Scott, Vint Cerf and Scott Burleigh for fruitful discussions on the custody transfer proposal and other aspects of the IPN architecture.

## References

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings ACM SIGCOMM 2003*, August 2003.

[2] S. Burleigh, A. Hooke, L. Torgerson, K. fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-tolerant networking: An approach to interplanetary internet," *IEEE Communications*, pp. 128–137, June 2003.

[3] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://citeseer.nj.nec.com/saltzer84endtoend.html

[4] D. D. Clark, "The design philosophy of the DARPA internet protocols," in *SIGCOMM*. ACM, Aug. 1988, pp. 106–114.

[5] D. Skeen, "Nonblocking commit protocols," in *SIGMOD*. ACM, 1981, pp. 133–142.

[6] A. Fox and E. Brewer, "Harvest, yield, and scalable tolerant systems," in *In Proceedings of HotOS-VII*, 1999.

[7] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Duke University, Tech. Rep. Technical Report CS-200006, 2000. [Online]. Available: http://www.cs.duke.edu/~vahdat/ps/epidemic.pdf

[8] P. Juang, H. Oki, Y. Wang, M. Martonosi, and L. P. D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," in *Proceedings of the Tenth ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002.

[9] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," November 2001, iCDCS-22.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *"Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)"*, 2002.

[11] P.Bonnet, J.Gehrke, and P.Seshadri, "Towards sensor database systems," in *Conference on Mobile Data Management*, January 2001.

[12] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, E. Travis, and H. Weiss, "Interplanetary internet (ipn): Architectural definition," May 2001. [Online]. Available: http://www.ipnsig.org/reports/memo-ipnrg-arch-00.pdf